Bin-summarise-smooth: A framework for visualising large data

Hadley Wickham

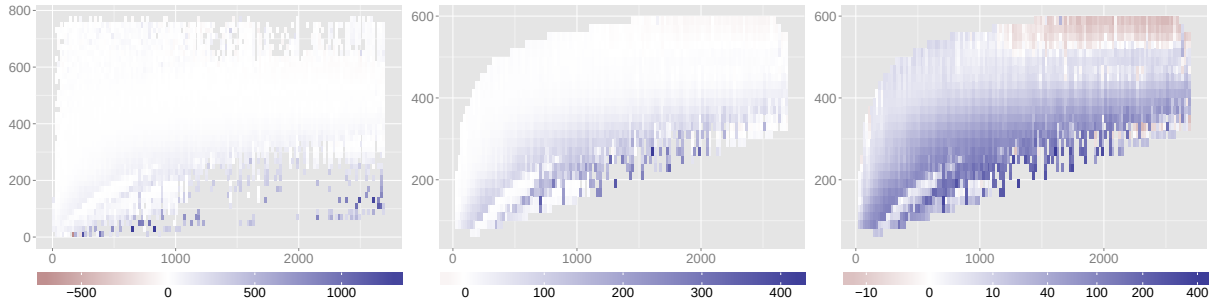

Fig. 1. Average delay (colour, in minutes) as a function of distance (x axis, in miles) and speed (y axis, in mph) for 76 million flights. The initial view (left) needs refinement to be useful: first we focus on the middle 99.5% of the data (centre) then transform average delay to shrink the impact of unusually high delays and focus on typical values (right). Flights with higher than average speeds (top-right) have shorter delays (red); more interestingly, a subset of shorter, slower flights (bottom-left) have average delays very close to 0 (white).

Abstract—
Visualising large data is challenging both perceptually and computationally: it is hard to know what to display and hard to efficiently display it once you know what you want. This paper proposes a framework that tackles both problems, based around a four step process of bin, summarise, smooth, and visualise. Binning and summarising efficiently (O(n)) condense the large raw data into a form suitable for display (recognising that there are ∼3 million pixels on a screen). Smoothing helps resolve problems from binning and summarising, and because it works on smaller, condensed datasets, it can make use of algorithms that are more statistically efficient even if computationally expensive. The paper is accompanied by a single-core in-memory reference implementation, and is readily extensible with parallel and out-of-memory techniques.

Index Terms—Big data, statistical graphics, kernel smoothing.

1 Introduction

As data grows ever larger, it is important that our ability to visualise it grows too. This paper presents a novel framework for visualising large data that is designed to be both computationally and statistically efficient, dealing with both the challenges of what to display for very large datasets and how to display it fast enough for interactive exploration. The insight that underlies this work is simple: the bottleneck when visualising large datasets is the number of pixels on a screen. At most we have around 3,000,000 pixels to use, and 1d summaries need only around 3,000 pixels. There is no point displaying more data than pixels, and rather than relying on the rendering engine to intelligently reduce the data, we develop summaries built on well-known statistical principles.

My work in R supplies many of the constraints that have guided this paper. The accompanying reference implementation is designed to be used by experienced analysts in conjunction with other data manipulation tools and statistical models. My goal was for users to be able to plot 10^8 observations in under 5s on commodity hardware. 10^8 doubles occupy a little less than 800 Mb, so about 20 vectors can be stored in 16 Gb of ram, with about 1 Gb left over. Five seconds is well above the threshold for direct manipulation, but is in line with how long other data manipulation and modelling functions take in R. Too much longer than 5s and it becomes tempting to give up waiting and switch to another task, which breaks flow and reduces productivity. To calibrate, 5s is about how long it takes to draw a regular scatterplot of 200,000 points in R, so I aim to be several orders of magnitude faster than existing work.

The framework involves four steps: binning, summarising, smoothing and visualising. Binning and summarising condense the large raw data to a summary on the same order of size as pixels on the screen. To be computational efficient, binning and summarising make some statistical sacrifices, but these can be compensated for with the smoothing step. Smoothing is generally computationally harder, but smoothing condensed data is fast and loses little statistical strength. The bin-summarise-smooth framework encompasses the most important 1d and 2d statistical graphics: histograms, frequency polygons and kernel density estimates (kdes) for 1d; and scatterplots, scatterplot smoothers and boxplots for 2d. It is readily extensible to new visualisations that use other summaries.

Section 3 discusses binning and summarising, focussing on the tension between computation and statistical concerns. Computationally, we want linear complexity and the ability to parallelise, but statistically, we want summaries that are resistant to unusual values. Section 4 discusses smoothing, focussing more on the statistical side, and shows how we can remedy some of the problems generated by the fast binning and summarising steps. Even once we've reduced the data to manageable size, visualisation of large data presents some special challenges. Section 5 discusses generally how to visualise the condensed datasets, and how to overcome problems with the outliers that are always present in large data.

The paper is accompanied by an open-source reference implementation in the form of the bigvis R [38] package. This is available from `http://github.com/hadley/bigvis` and is described in Section 6. The reference implementation focusses on in-memory,

• *Hadley Wickham is Chief Scientist at RStudio. E-mail: h.wickham@gmail.com.*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 27 September 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

single-core summaries of continuous data with an eye to producing static graphics. But these are not limitations of the framework, and I discuss avenues for future work in Section 7.

To illustrate the framework I include figures generated from the flight on-time performance data made available by the Bureau of Transportation Statistics[1]. I use flight performance data for all domestic US flights from 2000 to 2011: ~78,000,000 flights in total. The complete dataset has 111 variables, but here we will explore just four: the flight distance (in miles), elapsed time (in minutes), average speed (in mph) and the arrival delay (in minutes). The data was mildly cleaned: negative times, speeds greater than 761 mph (the speed of sound), and distances greater than 2724 miles (the longest flight in the continental US, SEA–MIA) were replaced with missing values. This affected ~1.8 million (2.4%) rows. The data, and code, needed to reproduce this paper and accompanying figures can be found at `http://github.com/hadley/bigvis`.

2 Related work

There is an extensive body of statistical research on kernel density estimation (kde, aka smoothed histograms) and smoothing, stretching back to the early 70's. Three good summaries of the work are [43, 4, 35]. [21, 49, 16] focus on the computational side, where much work occurred in the early 90's as statisticians transitioned their algorithms from mainframes to PCs. This work focusses mostly on the statistical challenges (asymptotic analysis), and is framed in the context of big data challenges of the time (1000s of points).

The statistical smoothing literature has had relatively little impact on infovis. [32] uses kernel density estimates, and provides a very fast GPU-based implementation, but fails to connect to the existing statistics literature and thus reinvents the wheel. Other techniques from infovis—footprint splatting [1, 57] and using transparency to deal with overplotting [26, 47]—can be framed as kernel density problems.

There are other approaches to large data: [22] discusses the general challenges, and proposes interaction as a general solution. Others have used distortion [29] or sampling [50, 3]. Distortion breaks down with high data density, as low density regions may be distant and the underlying spatial relationships become severely distorted. Sampling can be effective, particularly if non-uniform methods are used to overweight unusual values, but to have confidence in the final results you must either look at multiple plots or very careful select tuning parameters.

[34] describes a strikingly similar framework to this paper, motivated by interactive web graphics for large data. It is complimentary to the bin-summarise-smooth framework: it focusses on interaction and high-performance parallel GPU computation within the browser, but does not explore summaries other than count, or explore the importance of a smoothing step.

3 Condense

The bin and summary steps condense the large original dataset into a smaller set of binned summaries. Figure 2 illustrates the principle with the distance variable. Each flight is put in one of 237 10-mile-wide bins, then each bin is collapsed to three summaries: the count, the average speed, and the standard deviation of the speed. In general, this process involves first assigning each observation to an integer bin, as described in Section 3.1; then reducing all observations in a bin to a handful of summary statistics, as described in Section 3.2.

3.1 Bin

Binning is an injective mapping from the real numbers to a fixed and finite set of integers. We use **fixed width binning**, which is extremely fast, easily extended from 1d to nd, and while statistically naïve, there is little evidence that variable binwidths do better.

Binning needs to be considered as a separate step, as it may be performed outside of the visualisation environment. For example, binning could be done in the database, reducing data transmission costs by half since integers need only 4 bits of storage, while doubles need 8.

[1] `http://transtats.bts.gov/Fields.asp?Table_ID=236`

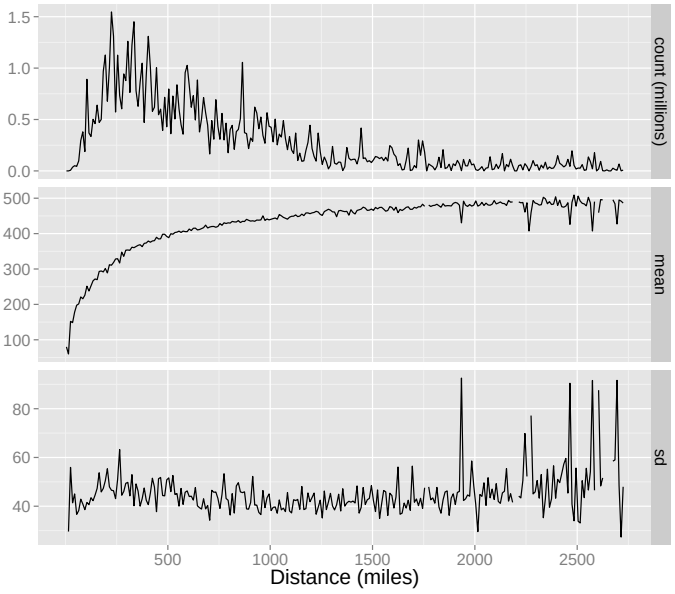

Fig. 2. Distance binned into 273 10-mile-wide bins, summarised with count, mean speed and standard deviation of speed. Note the varying scales on the y-axis, and the breaks in the line at extreme distances caused by missing data.

3.1.1 Computation

Fixed width binning is parametrised by two variables, the origin (the position of the first bin) and the width. The computation is simple:

$$\left\lfloor \frac{x - \text{origin}}{\text{width}} \right\rfloor + 1 \tag{1}$$

Bins are 1-indexed, reserving bin 0 for missing values and values smaller than the origin.

3.1.2 Extension to nd

Fixed width binning can be easily extended to multiple dimensions. You first bin each dimension, producing a vector of integers $(x_1, x_2, ..., x_m)$. It is then straightforward to devise a bijective map between this vector and a single integer, given that we know the largest bin in each dimension. If we have m dimensions, each taking possible values $0, 1, \ldots, n_m$, then we can collapse the vector of integers into a single integer using this formula:

$$\begin{aligned} &= x_1 + x_2 \cdot n_1 + x_3 \cdot n_1 \cdot n_2 + \cdots + x_m \cdot \Pi_{i=1}^{n-1} n_i \\ &= x_1 + n_1 \cdot (x_2 + n_2 \cdot (x_3 + \cdots (x_m)) \end{aligned} \tag{2}$$

It is easy to see how this works if each dimension has ten bins. For example, to project 3d bin $(5,0,4)$ into 1d, we compute $5 + 0 \cdot 10 + 4 \cdot 100 = 4 + 10 \cdot (0 + 10 \cdot 5) = 405$. Given a single integer, we can find the vector of m original bins by reversing the transformation, peeling off the integer remainder after dividing by 10. For example, 1d bin 356 corresponds to 3d bin $(6,5,3)$.

This function is a monotone minimal perfect hash, and highly efficient hashmap variants are available that make use of its special properties [2]. For example, because the hash is perfect, we can eliminate the equality comparison that is usually needed after a candidate has been found with the hashed value. Even if this data structure is not used (as in the reference implementation), it easy to efficiently summarise bins in high-dimensions using standard data structures: a vector if most bins have data in them, a hashmap if not.

The challenges of working with nd summaries are typically perceptual, rather than computational. Figure 3 shows a 2d summary of distance and speed: even moving from 1d to 2d makes it significantly harder to accurately perceive count [11].

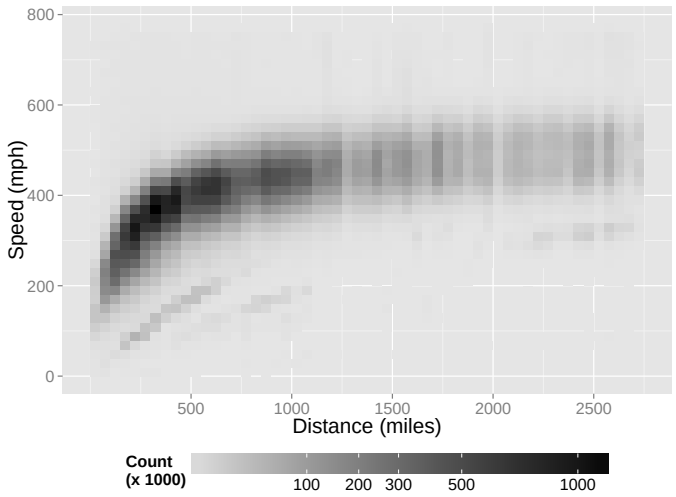

Fig. 3. Distance and speed summarised with count. Note that count scale has been transformed slightly, as described in Section 5.3.2, to draw attention to regions with fewer counts.

3.1.3 Statistical limitations

Ideally, we would like bigger bins in regions with few data points because the error of the summary is typically $\Theta(1/\sqrt{n})$. For example, in Figure 2 we would like to have bigger bins for larger distances since the counts are small and the estimates of mean and standard deviation are more variable than we would like. However, there is little evidence that varying binwidths leads to asymptotically lower error [46]. Varying bin widths should provide a better approximation to the underlying data, but the optimisation problem is so much more challenging that any potential improvements are lost. Instead of resolving these issues with a more sophisticated binning algorithm, we will fix them in the later smoothing step.

3.2 Summarise

Once each of the n original data points has been placed into one of m integer bins ($m \ll n$), the next step is to collapse the points in each bin into a small number of summary statistics. Picking useful summary statistics requires balancing between computational efficiency and statistical robustness.

3.2.1 Computational efficiency

Gray et al. [19] provide a useful classification for summary statistics. A summary is:

- **distributive** if it can be computed using a single element of interim storage and summaries from subgroups can be combined. This includes count, sum, min, and max.
- **algebraic** if it is a combination of a fixed number of distributive statistics. This includes the mean (count + sum), standard deviation (count + sum + sum of squares) and higher moments like skewness and kurtosis.
- **holistic** if it requires interim storage that grows with the input data. This includes quantiles (like the median), count of distinct elements or the most common value.

Algebraic and distributive statistics are important because results from subgroups can easily be combined. This has two benefits: it makes parallelisation trivial, and it supports a tiered approach to exploration. For example, if you have 100 million observations, you might first finely bin into 100,000 bins. Then for any specific 1d plot, you rebin or subset the fine bins rather than the original data. This tiered approach is particularly useful for interactive visualisations; the fine binning can be done upfront when the visualisation is created, then binwidths and plot limits can be modified interactively.

It is often possible to convert a summary from holistic to algebraic by taking an approximation. For example, the count of distinct values can be approximated with the hyperloglog algorithm [18], the median with the remedian [39], and other quantiles with other methods [17, 25, 33]. Others have proposed general methods for approximating any holistic summary [8].

The mean, standard deviation and higher moments can all be computed in a single pass, taking $O(n)$ time and $O(1)$ memory. Some care is needed as naive implementations (e.g. computing the variance as $\sum_i^n x_i^2/n - (\sum_i^n x_i/n)^2$) can suffer from severe numerical problems, but better algorithms are well known [51]. The median also takes $O(n)$ time (using the quick-select algorithm), but needs $O(n)$ memory: there is no way to compute the median without storing at least half of the data, and given the median of two subgroups, no way to compute the median of the full dataset.

3.2.2 Statistical robustness

There is an interesting tension between the mean and the median: the median is much more robust to unusual values than the mean, but requires unbounded memory. A useful way to look at the robustness of a statistic is the **breakdown point**. The breakdown point of a summary is the proportion of observations that an attacker needs to control before they can arbitrarily influence the resulting summary. It is 0 for the mean: if you can influence one value, you can force the mean to be any value you like. The breakdown point for the median is 0.5: you have to taint 50% of the observations before you can arbitrarily change the median. The mean is computationally desirable, but is less statistically desirable since just one flawed value can arbitrarily taint the summary. This is a general problem: the easiest summary statistics to compute are also the least robust, while robust statistics are usually holistic.

Even if you do use robust statistics, you are only protected from scattered outliers, not a radical mismatch between the data and the summary. For example, a single measure of central tendency (mean or median) will never be a good summary if the data has multiple modes. Compare Figures 2 and 3: for shorter flights, there appears to be multiple modes of speed for a given distance and so the mean is not a good summary. Visualisation must be iterative: you can not collapse a variable to a single number until you have some confidence that the summary is not throwing away important information. In practice, users may need to develop their own summary statistics for the peculiarities of their data; the ones discussed here should provide a good start for general use.

3.2.3 Higher dimensions

There is no reason to limit ourselves to only 1d summary functions. 2d summaries like the correlation may also be interesting. All statistics from a linear model can be computed in $O(n)$ in time and $O(1)$ in space [37], and thus suggest a fruitful ground for generalisations to higher dimensions. Other quickly computed 2d summaries are also of interest; scagnostics [55] are an obvious place to start.

4 Smooth

Smoothing is an important step because it allows us to resolve problems with excessive variability in the summaries. This variability may arise because the bin size is too small, or because there are unusual values in the bin. Either way, smoothing makes it possible to use fast statistics with low breakdown points instead of slow and robust summaries. Figure 4 shows the results of smoothing Figure 2: much of the small-scale variation has been smoothed away, making it easier to focus on the broad trends. There is some suggestion that the standard deviation of speed is lowest for distances of 1000–1500 miles, and rises for both smaller and large distances. This is much harder to see in Figure 2.

There are many approaches to smoothing, but we use a family of kernel based methods, because they:

- are simple and efficient to compute,

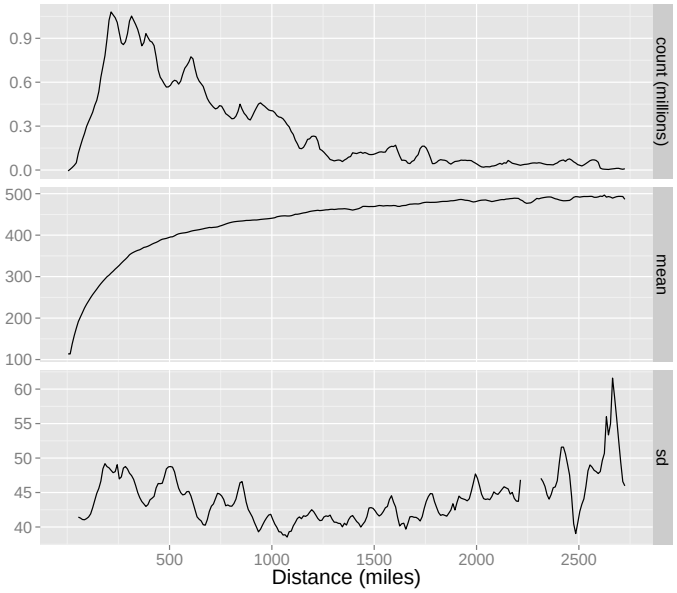

Fig. 4. The same underlying data from Figure 2, but smoothed with a bandwidth of 50. This removes much of the uninteresting variation while preserving the main trends.

- have a single parameter, the **bandwidth**, that controls the amount of smoothing,
- work just as well when applied to binned data [49],
- are approximately equivalent to other more complicated types of smoothing [44],
- form the heart of many existing statistical visualisations such as the kernel density plot [43], average shifted histogram [41] and loess [9].

Ideally, we would smooth the raw data, but it is too computationally expensive. Smoothing the binned summaries gives results that are visually very close to smoothing the original data, yet takes much less time.

4.1 How it works

Figure 5 illustrates the progression of smoothing methods from fast and crude to sophisticated and slower. The simplest smoothing method (top) is the binned mean, where we divide the data in bins and compute the mean of each bin. This is simple, but is locally constant and not very smooth. The next step up in complexity is the running (or nearest neighbour) mean where we average the five nearest points at each location. This is a considerable improvement, but is still rather jagged.

The remaining three types use a simple idea: we want to not only use the neighbouring points, but we want to weight them according to their distance from the target point. In statistics, the weighting function is traditionally called a **kernel**. There are a huge variety of kernels, but there is little evidence to suggest that the precise form of the kernel is important [10]. Gaussian kernels are common, but I use the triweight, $K(x) = (1 - |x|^3)^2 I_{|x|<1}$, because it is bounded and simple (evaluation of this function is $\sim$10$\times$ faster than the Gaussian).

At each bin i we have x_i the centre of the bin; y_i, the summary statistic; and w_i, the number of observations in the bin. To predict a smooth estimate at position j, we first compute the kernel weights for each location $k_i = K\left(\frac{x_j - x_i}{h}\right)$. The parameter h is called the bandwidth, and controls the degree of smoothing: larger h's include more neighbours and produce a smoother final curve. Because of the form of the triweight kernel, any observation more than h away from x_j will not contribute to the smoothed value, thus enabling efficient computation.

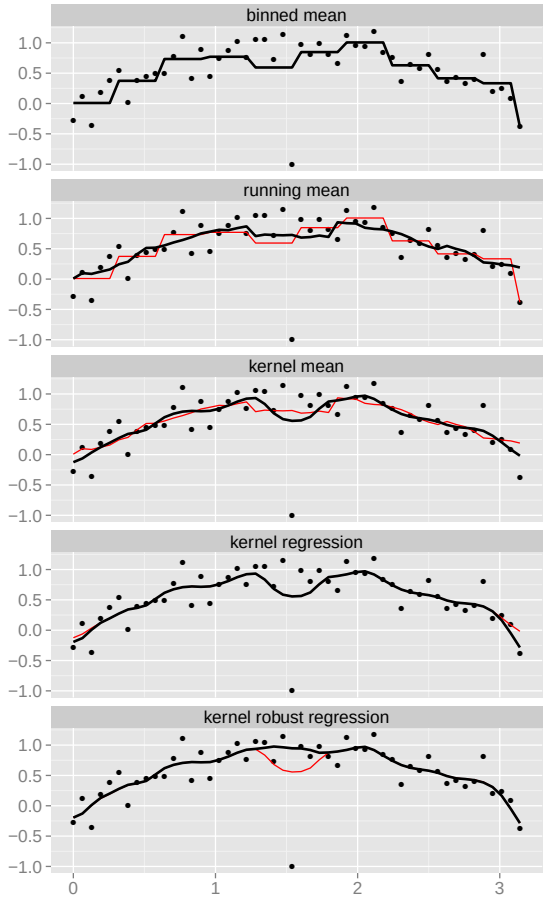

Fig. 5. Five types of smoothing on an artificial dataset generated with $\sin(x)$ on $[0, \pi]$, with random normal noise with $\sigma = 0.2$, and an outlier at $\pi/2$. Smooths are arranged from simplest (top, binned) to most accurate (bottom, robust local regression). To aid comparison each smooth is shown twice, prominently with a thick black line, and then repeated on the next plot in red to make comparison easier. The subtlest difference is between the kernel mean and kernel regression: look closely at the boundaries.

There are three kernel techniques: kernel means (aka Nadaraya-Watston smoothing), kernel regression (aka local regression) and robust kernel regression (aka loess). These make a trade-off between performance and quality. While closely related, these methods developed in different parts of statistics at different times, and the terminology is often inconsistent. [10] provides a good historical overview. To compute each smooths, we simply take the standard statistical technique (mean, regression or robust regression) and apply it to each sample location with weights $w_i \cdot k_i$.

The kernel (weighted) mean is fastest to compute but suffers from bias on the boundaries, because the neighbours only lie on one side. The kernel (weighted) regression overcomes this problem by effectively using a first-degree Taylor approximation. In Figure 5, you can see that the kernel mean and kernel regression are coincident everywhere except near the boundaries. Higher-order approximations can be used by fitting higher-order polynomials in the model, but there seems to be little additional benefit in practice [10].

Finally, the robust kernel regression iteratively down-weights the effect of points far away from the curve, and reduces the impact of unusual points at the cost of increased computation time (typically the number of iterations is fixed, so it is a constant factor slower than regular regression). There are many ways to implement robust regression, but the procedure used by loess [9] is simple, computational tractable and performs well in practice. The key advantage of a robust smooth can be seen in Figure 5: it is the only smoother unaffected by the un-

usually low point at $\pi/2$.

4.2 Higher-dimensions and performance

The three kernel techniques are readily extended to higher dimensions: for example, instead of computing a 1d weighted mean, you compute a nd weighted mean. Kernel means are particularly easy to efficiently extend to higher dimensions because they are a convolution. Due to the associative nature of convolution, it is possible to replace a d-dimensional smooth with a sequence of d 1d smooths. This is an important optimisation because it converts an $O(n^d)$ process to $O(nd)$. You can also perform kernel regression through a sequence of 1d regressions, but it will only be exact if the underlying grid of values is uncorrelated. Robust kernel regression can not be approximated in this way, and must be performed exactly.

Given that the kernel mean is a convolution, it's natural to wonder if a discrete fast Fourier transform (dFFT) would be useful. My experience is that it is not helpful in this case: it requires more complicated code to deal with the periodic nature of the dFFT; it is less flexible if you want to predict smooth values at only a subset of locations; and it provides only negligible performance improvements in 1d [49].

4.3 Automatic bandwidth selection

Kernel smoothing introduces a tuning parameter which controls the degree of smoothing: the bandwidth. There is much statistical research on how to pick the "best" bandwidth but it is typically framed in the content of reducing integrated mean squared error, and it is not obvious that this corresponds with what we want for visualisation [12]. Following [36], I take a pragmatic approach and use leave-one-out cross-validation (LOOCV) [14] to provide a starting point for the user. In a realistic data analysis scenario, this would be supplemented with interactive controls so the user could explore findings at different levels of resolution.

The intuition behind LOOCV is simple: we compare the actual statistic at each location with its smoothed prediction computed without that observation. The observation level errors are summarised with the root mean squared error (rmse): $\sqrt{\sum(y_i - \hat{y}_i)^2/n}$, and we look for the bandwidth that has the smallest rmse. We can visually explore the rmse across a range of possible grid values, or use standard numerical optimisation (like L-BFGS-B [7]) to find a minima. Figure 6 shows the LOOCV rmse error for the three variables in Figure 2. It demonstrates the challenges of relying on numerical optimisation alone: mean and sd have a number of local minima, while count is very flat.

With the bounded tricube kernel, LOOCV is straightforward to implement efficiently in $O(mb)$, where b is the number of bins covered by the kernel ($b \ll m$). [16] suggests incremental algorithms that may reduce computation time still further.

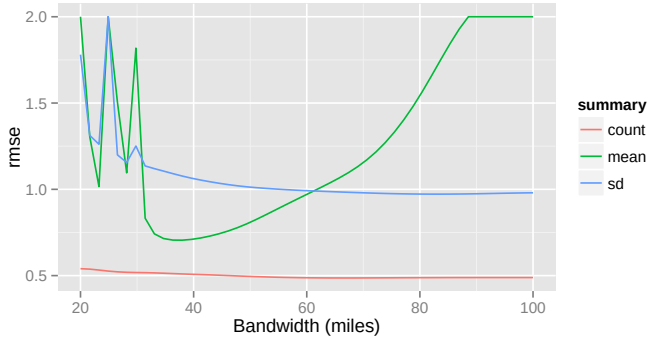

Fig. 6. A leave one out cross-validated estimate of the root-mean-squared-error associated with smoothing the count, mean, and sd from Figure 2 with varying bandwidths. Rmse has been capped at 2 to focus on the regions with lower errors.

4.4 Varying bandwidths

Intuitively, it seems like we should be able to do better with an adaptive binwidth. We could use smaller bins where there is more data, or where the underlying curve is especially wiggly. There are many approaches documented in the literature [46, 6, 40, 24]. We follow the simple, but performant, approach outlined in [15]: to find a per-location h, we divide the data into $m/(10 \cdot \log(m))$ pieces, estimate the optimal bandwidth for each piece separately, then smooth the estimates.

Figure 7 illustrates the process for a challenging function proposed by [15]. It demonstrates the effectiveness of this approach: a fixed bandwidth will either over-smooth on the left or under-smooth on the right, whereas a variable bandwidth can adapt to provide visually equivalent smoothness across the entire range of the function.

5 Visualise

Once we have binned, summarised and smoothed the raw data, it is relatively straightforward to visualise it. When discussing visualisations, it's useful to distinguish between the binned variables from the original data set, and the new variables created by the summaries. To make that clear we will adopt the convention that a (n,m)-d dataset or visualisation represents a dataset with n binned variables and m summary variables. For example, Figure 2 shows 3 $(1,3)$-d plots (distance, count + mean + sd), Figure 3 shows a $(2,1)$-d plot (distance + speed, count), and Figure 1 is a $(2,1)$-d visualisation produced from a $(2,2)$-d dataset (distance + speed, count + mean).

Generally, we are interested in plots where both m and n are one or more. A $(0,1)$ plot would just display the distribution of the summary statistic across all bins, and a $(m,0)$ plot would just show where the bins occurred. A $(0,2)$ plot might be of interest, because you could compare summary statistics within each bin (e.g. you might be interested in bins with high mean and low standard deviation), but because it does not show the original context, we will not consider it further. Section 5.1 discusses $(1,1)$-d plots, and Section 5.2 $(2,1)$-d plots. Section 5.3 discusses general combinatoric methods for combining low-d plots to deal with n-d data.

As the size of data grows, the probability of finding very unusual values also increases. In practice, I found many large data visualisations look like the initial plot in Figure 1, with most data concentrated in a small area. We have two tools to deal with this: peeling for outliers in space, Section 5.3.1, and the modulus transformation for outliers in colour and other aesthetics, Section 5.3.2.

Finally, it is important to preserve missing values: this is not an issue just for big data, but is worth mentioning here. Section 5.4 discusses the importance of missing data, and why the plots you have seen so far are potentially misleading.

5.1 $(1,1)$-d plots

$(1,1)$-d plot are simple to represent with a line. If the summary statistic is a count, this produces a frequency polygon [42]. This is an improvement on the traditional histogram as it is easier to overlay multiple frequency polygons in one plot, and it unifies the display of counts and other summaries.

If the statistic is not a count, we also need some way of calibrating for the uncertainty of the estimate. This is important because we don't draw strong conclusions from areas with weak data support. We have already seen one option in Figure 2: display both the count and the summary in adjacent plots. Figure 8 shows two other options, mapping either the count or relative error to the colour of the line. Using the count shows us where the majority of the data lie, but a measurement of error is more important for calibrating our expectation of uncertainty. Using relative error makes it possible to use a common scale across all plots, where errors above a certain threshold (here 10%) blend into the plot background. In this example, apart from the first line segment, all errors are less than 0.3%.

Estimates of error can be found through asymptotic approximations, like the central limit theorem (CLT), or computationally with the bootstrap [14]. For example, the standard error of the mean given by the CLT is $\sigma/\sqrt{n}$, and this approximation is known to be good for

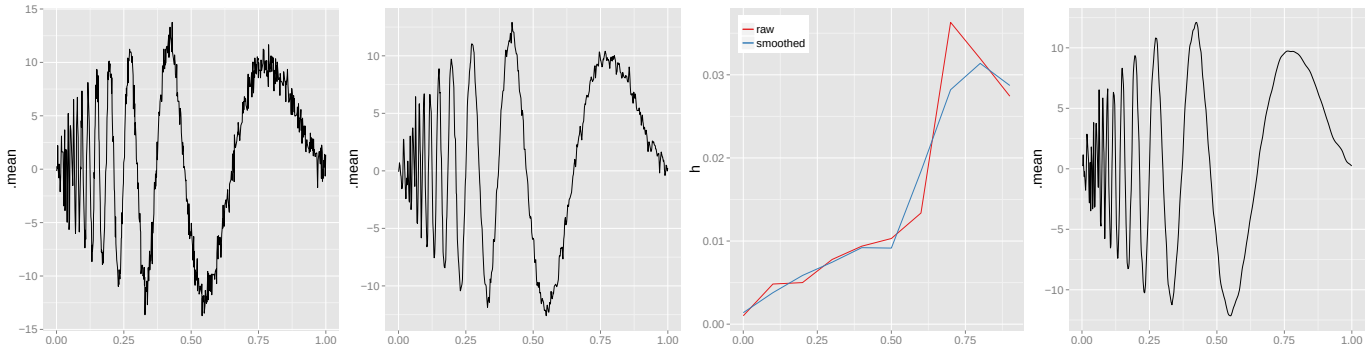

Fig. 7. (Far left) A binned summary from a challenging function to smooth with fixed bandwidth [15]. (Left) Data smoothed using single best bandwidth. (Right) Optimal bandwidth estimated on 10 separate pieces. (Far right) Data smoothed using individual bandwidth at each location.

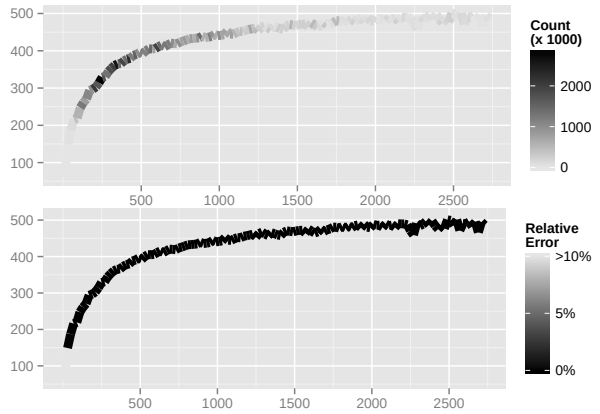

Fig. 8. Two ways to calibrate the uncertainty associated with a $(1,1)$-d plot of mean speed on distance in 20-mile-wide bins. From top-to-bottom: the number of observations at each location, absolute error, or relative error.

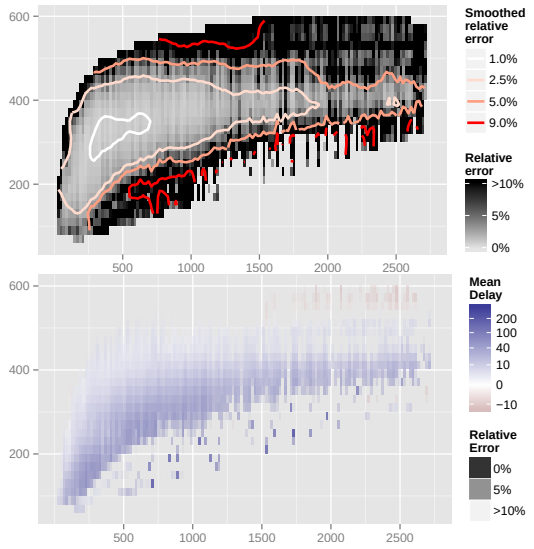

Fig. 9. Three ways to display the uncertainty associated with a $(2,1)$-d plot of speed, distance, and average delay, as in Figure 1. (Top) A tile plot and smoothed contours of relative error. (Bottom) Relative error mapped to transparency.

most underlying distributions as long as n is relatively large (a common rule of thumb is 30). The standard error of other summaries is typically more difficult to derive, but it is usually easy to approximate with the boostrap. The disadvantage of the boostrap is that it requires multiple passes through the data, selecting a different random subset each time.

5.2 $(2,1)$-d plots

$(2,1)$-d plots require a display like Figure 1, which is often called a heatmap [56] (although that name is sometimes reserved for plots of categorical data), or a a tile plot. Other alternatives are contour plots (particularly for smoothed data), or the many other cartographic techniques for displaying surfaces [30].

Calibrating for uncertainty is more challenging in 2d. Figure 9 demonstrates two ways of showing the error associated with the estimates in Figure 1. The top figure shows a tile plot of relative error overlaid with a smoothed contour plot. The bottom figure maps transparency to relative error so that regions with high errors are hard to see. None of these approaches are perfect: a separate tile plot of relative error is hard to integrate with the other summary; overlaying a contour plot is challenging because the underlying surface is not very smooth and there are patches of unusual values; and using transparency gives a qualitative, not quantitative, impression of uncertainty. Better approaches likely require interactivity, as discussed in the following section.

5.3 (n,m)-d plots

Extending these graphics to higher dimensions is challenging, but two general avenues of attack exist: small multiples and interaction.

Small multiples (aka faceting) can be used to display low-d slices through a high-d dataset. The product plots [54] framework explores this in detail for count summaries of categorical data. It can be readily extended to continuous data, because binned continuous variables can be treated the same way as categorical variables. There are two additonal constraints: tiles (treemaps) are no longer appropriate because they do not preserve ordering, and initial variables need to be relatively coarsely binned to ensure there is sufficient space to display variables lower in the hierarchy.

An alternative approach is to use interaction. Linked brushing provides an effective way to connecting multiple low-dimensional displays to gain high-dimensional insight [34, 45]. This can also reduce the data summary burden [22].

5.3.1 Peeling

A simple approach to deal with spatial outliers is to remove the least populated outer bins. I call this approach peeling, and have implemented it by progressively removing the smallest counts on the convex hull of the data. Figure 1 illustrates the utility of this technique: even peeling off a tiny fraction of the data (0.5%) yields a substantially improved plot. Some small amount of peeling (often $< 1\%$) seems to improve most 2d plots, drawing the focus to the bulk of the data. For plots of non-count summaries, this also has the additional benefit of reducing outliers in the summary dimension.

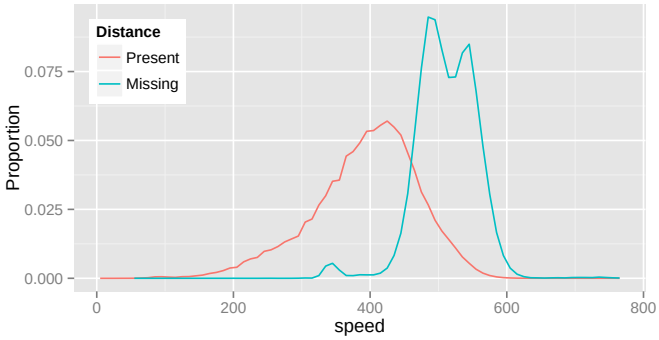

Fig. 10. The distribution of speed depending on whether distance is present or missing.

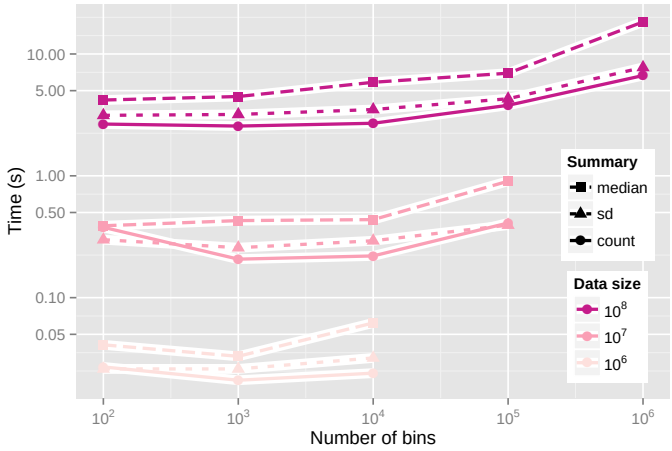

Fig. 11. Benchmarking the bigvis package by varying the input size (colour), number of bins (x-axis) and summary statistic (shape and line type). Note that both x and y axes are logged.

Rather than simply throwing outliers away, it is better to partition the analysis to look separately at the common and at the unusual. Outliers are often interesting!

5.3.2 Modulus transformation

Removing bins with small counts will sometime remove the most outlying values in the summary dimension, but often many outliers will remain. It's useful to have a flexible technique to down-weight the visual impact of these unusual values, without removing them altogether. I suggest using the modulus transformation [27], which provides a flexible tool to shrink or expand the tails of a distribution. Figure 1 demonstrates the importance of a mild transformation to avoid outlier-induced distortion of the colour scale.

The modulus transformation generalises the Box-Cox transformation [5] to deal with both positive and negative data:

$$\begin{cases} \text{sgn}(x) \cdot \log(|x|+1) & \text{if } \lambda = 0, \\ \text{sgn}(x) \cdot \frac{(|x|+1)^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0. \end{cases} \quad (3)$$

The λ parameter controls the strength of the transformation, with the strongest transformation at $\lambda = 0$.

The modulus transformation is particularly useful in conjunction with interactivity, as it allows the user to dynamically focus on important parts of the distribution. This provides a smooth alternative to discontinuous scales that force the inclusion of certain values [28].

5.4 Missing values

An important design principle is that missing values should never be omitted without user intervention: they should always be preserved along the entire route from raw data to final visualisation. It should be possible to remove them, but it must be a deliberate decision by the analyst: you never want to distort a visualisation by silently leaving off data. This an important principle of MANET [48], and is essential for real data which often have missing values, either in the raw data, or introduced during data cleaning. This principle is generally simple to implement, but must be carried through all other components of the framework: missing values must be preserved during binning, summarising and smoothing.

That said, none of the plots in this paper actually preserve missing values. Figure 10 shows why this is a bad idea: the distribution of speed is very different depending on whether or not distance is missing. Distance is clearly not missing at random, so ignoring missing values may lead to skewed conclusions.

6 The `bigvis` package

The bigvis package provides a reference implementation of the bin-summarise-smooth framework with an open-source R package available from `http://github.com/hadley/bigvis`. It has two main parts: a high-performance implementation of the key algorithms written in C++, and a user-friendly wrapper that supports exploratory data analysis written in R. The key to bridging the two pieces is the Rcpp package [13] which provides an easy way to access R's internal data structures with a clean C++ API.

The C++ component makes use of templated functions to avoid the cost of virtual method lookup (this is small, but adds up with potentially 100's of millions of invocations). This adds an additional challenge when connecting R and C++ because templated functions are specialised at compile-time, but the R code needs to call them dynamically at run-time. To work around this problem, a small code generator produces all specialised versions of templated functions. This is somewhat inelegant, but unavoidable when coupling programming languages with such different semantics.

The bigvis API is designed to make visualising large datasets familiar to R users. It implements methods for the `autoplot` generic function, providing default visualisations designed with the principles above, and also cleanly interfaces with other plotting tools in R, like ggplot2 [52, 53]. This makes it easy to build custom graphics based on the condensed and smoothed data if the default graphics are not sufficient. The figures in this paper were all drawn with ggplot2, and are heavily customised for maximal clarity.

6.1 Benchmarks

Figure 11 provides basic benchmarks for the bigvis package, over a range of data input sizes, bins and summary statistics. Timings were performed on a 2.6 GHz Intel Core i7, 15" MacBook retina with 16 Gb ram and a solid state hard drive. The benchmarks assume the data are already in memory: loading data from R's binary on-disk serialisation takes around 2s for 10^8 observations.

Computing time grows linearly with input size and slowly with number of bins (only conditions where there would be 10 or more observations in each bin are shown). There is relatively little variation between the count, standard deviation and mean. The three summaries are all $O(n)$, but the standard deviation has a higher constant than the mean, and the median is slowest because it requires a complete additional copy of the data.

These timings are lower bounds on potential performance: I have been programming in C++ for less than six months so the C++ code is by no means expert and uses only a smattering of advanced C++ functions. There are likely to be considerable opportunities for further optimisation.

7 Future work

The bin-summarise-smooth framework provides a flexible toolkit for the visualisation of large datasets, designed around the pixels-on-screen bottleneck. It balances computational and statistical concerns to create a framework that scales well with data size, without sacrificing the fidelity of the underlying data.

While the reference implementation is in-memory and single-core, parallelisation and out-of-memory datastores are obvious areas for future work. We expect that fine pre-binning with simple summaries may be done in an existing datastore; the grouping algorithm is trivial to implement in SQL, and most databases already provide simple summary statistics like the mean. Other systems, e.g. madlibs [23], may provide richer summaries making grouping and summarising in the database even more appealing. Column store databases [31] are a natural fit with statistical data, and may provide efficient out-of-memory back ends. Multicore and GPU implementations providing exciting avenues to extend this framework to deal with a billion points and beyond.

This paper has focussed on continuous data, but it is straightforward to extend to categorical variables that take on relatively few values. Such variables can be trivially mapped to integers, and smoothing is usually not appropriate. The challenge typically arises during visualisation, where some seriation [20] is often needed to draw out important patterns. An additional challenge is when the number of categories grows with the size of the data: if there are more categories than pixels, which categories do you collapse together?

Finally, it's important to note that visualisation is only one piece of the data analysis puzzle: we also need tools for transforming and modelling big data. Visualisation is excellent for raising and refining questions, but does not scale well: a human needs to look at each plot. Statistical models and other algorithms scale well, but never reveal what you fundamentally don't expect. Tools for visualisation must interconnect with tools for modelling, so practitioners can take advantages of the strengths of both toolsets. This is one reason I work in R; practitioners can combine independent visualisation, modelling and manipulation tools to solve real problems.

Acknowledgments

I'd like to thank Yue Hu who coded up the initial prototypes, as well as JJ Allaire, Dirk Eddelbuettel, Romain Francois and Carlos Scheidegger who helped me learn C++ and answered many of my dumb questions. Early versions of this work were generously sponsored by Revolution Analytics and Google.

References

[1] B. G. Becker. Volume rendering for relational data. In *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis '97)*, pages 87–90, 1997.

[2] D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna. Monotone minimal perfect hashing: searching a sorted table with o (1) accesses. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 785–794. Society for Industrial and Applied Mathematics, 2009.

[3] E. Bertini and G. Santucci. Give chance a chance: modeling density to enhance scatter plot quality through random data sampling. *Information Visualization*, 5(2):95–110, 2006.

[4] A. W. Bowman and A. Azzalini. *Applied smoothing techniques for data analysis*. Oxford University Press, 1997.

[5] G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[6] M. Brockmann, T. Gasser, and E. Herrmann. Locally adaptive bandwidth choice for kernel regression estimators. *Journal of the American Statistical Association*, 88(424):1302–1309, 1993.

[7] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[8] A. Christmann, I. Steinwart, and M. Hubert. Robust learning from bites for data mining. *Computational statistics & data analysis*, 52(1):347–361, 2007.

[9] W. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, pages 829–836, 1979.

[10] W. S. Cleveland and C. Loader. Smoothing by local regression: Principles and methods. In W. Härdle and M. G. Schimek, editors, *Statistical theory and computational aspects of smoothing*, pages 10–49. Springer, New York, 1996.

[11] W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[12] L. Denby and C. Mallows. Variations on the histogram. *Journal of Computational and Graphical Statistics*, 18(1):21–31, 2009.

[13] D. Eddelbuettel and R. Francois. Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40(8):1–18, 4 2011.

[14] B. Efron and G. Gong. A leisurely look at the bootstrap, the jackknife, and cross-validation. *The American Statistician*, 37(1):36–48, 1983.

[15] J. Fan and I. Gijbels. Data-driven bandwidth selection in local polynomial fitting: variable bandwidth and spatial adaptation. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 371–394, 1995.

[16] J. Fan and J. S. Marron. Fast implementations of nonparametric curve estimators. *Journal of Computational and Graphical Statistics*, 3(1):35–56, 1994.

[17] M. Finkelstein and S. Scheinberg. The storage complexity of some estimators of the median. *Applied Mathematics and Computation*, 60:15–24, 1994.

[18] P. Flajolet, É. Fusy, O. Gandouet, F. Meunier, et al. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Conference on Analysis of Algorithms*, volume 7, pages 127–145, 2007.

[19] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–53, 1997.

[20] M. Hahsler, K. Hornik, and C. Buchta. Getting things in order: An introduction to the r package seriation. *Journal of Statistical Software*, 25(3), 2008.

[21] W. Härdle and D. Scott. Smoothing in low and high dimensions by weighted averaging using rounded points. *Computational Statistics*, 7:97–128, 1992.

[22] J. Heer and S. Kandel. Interactive analysis of big data. *XRDS: Crossroads, The ACM Magazine for Students*, 19(1):50–54, 2012.

[23] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, et al. The madlib analytics library: or mad skills, the sql. *Proceedings of the VLDB Endowment*, 5(12):1700–1711, 2012.

[24] E. Herrmann. Local bandwidth choice in kernel regression estimation. *Journal of Computational and Graphical Statistics*, 6(1):35–54, 1997.

[25] C. Hurley and R. Modarres. Low-storage quantile estimation. *Computational Statistics*, 10:311–325, 1995.

[26] J. Johansson, P. Ljung, M. Jern, and M. Cooper. Revealing structure within clustered parallel coordinates displays. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 125–132. IEEE, 2005.

[27] J. John and N. Draper. An alternative family of transformations. *Applied Statistics*, pages 190–197, 1980.

[28] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 547–554. ACM, 2012.

[29] D. A. Keim, M. C. Hao, U. Dayal, H. Janetzko, and P. Bak. Generalized scatter plots. *Information Visualization*, 9(4):301–311, 2010.

[30] P. Kennelly. GIS applications to historical cartographic methods to improve the understanding and visualization of contours. *Journal of Geoscience Education*, 50(4):428–436, 2002.

[31] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. In *Proceedings of International Conference on Very Large Data Bases 2011 (VLDB)*, pages 585 – 597, 2011.

[32] O. D. Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 171–178. IEEE, 2011.

[33] J. C. Liechty, D. K. J. Lin, and J. P. Mcdermott. Single-pass low-storage arbitrary quantile estimation for massive datasets. *Statistics and Computing*, 13:91–100, 2003.

[34] Z. L. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum (Proc. EuroVis)*, 2013.

[35] C. Loader. *Local regression and likelihood*. Springer, New York, 1999.

[36] C. R. Loader. Bandwidth selection: classical or plug-in? *The Annals of Statistics*, 27(2):415–438, 1999.

[37] A. J. Miller. Algorithm as 274: Least squares routines to supplement those of gentleman. *Applied Statistics*, pages 458–478, 1992.

[38] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[39] P. J. Rousseeuw and G. W. Bassett, Jr. The remedian: A robust averaging method for large data sets. *Journal of the American Statistical Association*, 85(409):97–104, 1990.

[40] W. R. Schucany. Adaptive bandwidth choice for kernel regression. *Journal of the American Statistical Association*, 90(430):535–540, 1995.

[41] D. W. Scott. Averaged shifted histograms: Effective nonparametric density estimators in several dimensions. *The Annals of Statistics*, 13:1024–1040, 1985.

[42] D. W. Scott. Frequency polygons: theory and application. *Journal of the American Statistical Association*, 80(390):348–354, 1985.

[43] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley Series in Probability and Statistics. Wiley, New York, 1992.

[44] B. W. Silverman. Spline smoothing: the equivalent variable kernel method. *The Annals of Statistics*, pages 898–916, 1984.

[45] D. F. Swayne, D. Temple Lang, A. Buja, and D. Cook. GGobi: Evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43:423–444, 2003.

[46] G. R. Terrell and D. W. Scott. Variable kernel density estimation. *The Annals of Statistics*, 20(3):1236–1265, 1992.

[47] A. Unwin, M. Theus, and H. Hofmann. *Graphics of Large Datasets*. Springer, 2006.

[48] A. R. Unwin, G. Hawkins, H. Hofmann, and B. Siegl. Interactive graphics for data sets with missing values - MANET. *Journal of Computational and Graphical Statistics*, 5(2):113–122, 1996.

[49] M. Wand. Fast computation of multivariate kernel estimators. *Journal of Computational and Graphical Statistics*, 3(4):433–445, 1994.

[50] E. J. Wegman. Huge data sets and the frontiers of computational feasibility. *Journal of Computational and Graphical Statistics*, 4:281–295, 1995.

[51] B. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[52] H. Wickham. *ggplot2: Elegant graphics for data analysis*. useR. Springer, July 2009.

[53] H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

[54] H. Wickham and H. Hofmann. Product plots. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis '11)*, 17(12):2223–2230, 2011.

[55] L. Wilkinson, A. Anand, and R. Grossman. Graph-theoretic scagnostics. In *IEEE Symposium on Information Visualization*, pages 157–164, 2005.

[56] L. Wilkinson and M. Friendly. The history of the cluster heat map. *The American Statistician*, 63(2):179–184, 2009.

[57] L. Yang. Visual exploration of large relational data sets through 3d projections and footprint splatting. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1460 –1471, 2003.